
MODULE *FlowRuleStore*

EXTENDS *Naturals, Sequences, FiniteSets, TLC*

The set of node identifiers

CONSTANT *Node*

The set of available device identifiers

CONSTANT *Device*

The set of available flows

CONSTANT *Flow*

The total number of buckets

CONSTANT *NumBuckets*

A constant value

CONSTANT *Nil*

Message type constants

CONSTANTS

DigestsRequest,
DigestsResponse,
BucketsRequest,
BucketsResponse,
BackupRequest,
BackupResponse

A sequence of messages for each node

VARIABLE *messages*

States (terms and mastership) for each node

VARIABLE *states*

The highest term for each device, used to ensure terms are monotonically increasing

VARIABLES *terms*

A queue of mastership changes for each device on each node

VARIABLE *masterships*

The last logical backup time for each bucket/device/node

VARIABLE *backups*

The local logical clock for each node

VARIABLE *clocks*

The flow buckets for each device on each node

VARIABLE *flows*

An implementation of lamport clocks for causal ordering of events

Increments the logical clock for the given node

$TickClock(n) \triangleq states' = [states \text{ EXCEPT } ![n].timestamp = states[n].timestamp + 1]$

Updates the logical clock for the given node using the given timestamp

$UpdateClock(n, t) \triangleq$
 $\vee \wedge states[n].timestamp < t$
 $\wedge states' = [states \text{ EXCEPT } ![n].timestamp = t + 1]$
 $\vee \wedge states[n].timestamp \geq t$
 $\wedge states' = [states \text{ EXCEPT } ![n].timestamp = states[n].timestamp + 1]$

Messages are modelled as queues for consistency with *TCP* semantics. Each node has a separate channel for all requests and responses. The logical clock is managed on each send/receive by attaching a timestamp to all outgoing messages and updating the node's clock on receive.

Returns a sequence with the head removed

$Pop(q) \triangleq SubSeq(q, 2, Len(q))$

Sends a request on the given node's channel

$SendMessage(n, m) \triangleq$
 $\wedge TickClock(n)$
 $\wedge LET \text{ message} \triangleq [m \text{ EXCEPT }!.clock = clocks'[n]]$
 $IN \text{ messages}' = [messages \text{ EXCEPT } ![n] = Append(messages[n], message)]$

Removes a message from the given node's channel

$ReceiveMessage(n, m) \triangleq$
 $\wedge UpdateClock(n, m.clock)$
 $\wedge messages' = [messages \text{ EXCEPT } ![n] = Pop(messages[n])]$

This section models flows arbitrarily added to/removed from each node

Returns the bucket *ID* for the given flow *ID* by hashing the flow *ID* to the number of buckets

$GetBucket(fid) \triangleq (fid \% NumBuckets) + 1$

Adds a flow 'f' to node 'n' if it believes itself to be the master

The given flow is hashed to the appropriate bucket within the device table on node 'n'.

$AddFlow(n, f) \triangleq$
 $\wedge states[n][f.did].master$
 $\wedge TickClock(n)$
 $\wedge flows' = [flows \text{ EXCEPT } ![n][f.did][GetBucket(f.fid)] = [$
 $\text{term} \mapsto states[n][f.did].term,$
 $\text{timestamp} \mapsto clocks'[n],$
 $\text{entries} \mapsto flows[n][f.did][GetBucket(f.fid)] \cup \{f\}]$
 $\wedge UNCHANGED \langle messages, states, terms, masterships \rangle$

Removes a flow 'f' from node 'n' if it believes itself to be the master

The given flow is hashed to the appropriate bucket within the device table on node 'n'.

$$\begin{aligned}
\text{RemoveFlow}(n, f) &\triangleq \\
&\wedge \text{states}[n][f.\text{did}].\text{master} \\
&\wedge \text{TickClock}(n) \\
&\wedge \text{Cardinality}(\text{flows}[n][f.\text{did}][f.\text{fid}\% \text{NumBuckets}] \cap \{f\}) = 1 \\
&\wedge \text{flows}' = [\text{flows} \text{ EXCEPT } ![n][f.\text{did}][\text{GetBucket}(f.\text{fid})] = [\\
&\quad \text{term} \quad \mapsto \text{states}[n][f.\text{did}].\text{term}, \\
&\quad \text{timestamp} \mapsto \text{clocks}'[n], \\
&\quad \text{entries} \quad \mapsto \text{flows}[n][f.\text{did}][\text{GetBucket}(f.\text{fid})] \setminus \{f\}]] \\
&\wedge \text{UNCHANGED } \langle \text{messages}, \text{states}, \text{terms}, \text{masterships} \rangle
\end{aligned}$$

Mastership terms are modelled as a queue of monotonically increasing term/master notifications. Each node has a separate notification queue, and mastership terms are added to all queues in the same order. This models the fact that different nodes can learn of mastership changes at different times, but each node sees terms increase with the same master for each term.

One significant difference from the spec and the implementation is that the spec does not use limited numbers of backup nodes. If a node is a master it considers all other nodes to be backups.

Adds mastership term 't' with master 'n' to the mastership queues for device 'd'

$$\begin{aligned}
\text{AddTerm}(n, d, t) &\triangleq \\
&\wedge t > \text{terms}[d] \\
&\wedge \text{masterships}' = [\text{masterships} \text{ EXCEPT } ![d] = [m \in \text{Node} \mapsto \\
&\quad \text{Append}(\text{masterships}[d][m], [node \mapsto n, term \mapsto t])]] \\
&\wedge \text{terms}' = [\text{terms} \text{ EXCEPT } ![d] = t] \\
&\wedge \text{UNCHANGED } \langle \text{messages}, \text{states}, \text{backups}, \text{flows} \rangle
\end{aligned}$$

Notifies node 'n' of mastership term 't' for device 'd'

$$\begin{aligned}
\text{LearnTerm}(n, d, t) &\triangleq \\
&\wedge \text{masterships}' = [\text{masterships} \text{ EXCEPT } ![d][n] = \text{Pop}(\text{masterships}[d][n])] \\
&\wedge \text{states}' = [\text{states} \text{ EXCEPT } ![n][d].\text{term} = t.\text{term}, ![n][d].\text{master} = t.\text{node} = n] \\
&\wedge \text{UNCHANGED } \langle \text{messages}, \text{terms}, \text{backups}, \text{flows} \rangle
\end{aligned}$$

This section models the replication protocol. The protocol includes a simple backup mechanism which uses logical clocks to determine when buckets need to be replicated. Additionally, an anti-entropy protocol is used to detect out-of-date buckets on backup nodes.

Sends a backup request for device 'd' bucket 'b' from node 'n' to node 'm' if the bucket has been updated since the last backup

$$\begin{aligned}
\text{Backup}(n, d, b, m) &\triangleq \\
&\wedge n \neq m \\
&\wedge \text{LET } \text{bucket} \triangleq \text{flows}[n][d][b] \\
&\quad \text{IN} \\
&\quad \wedge \text{backups}[n][d][b][m] < \text{bucket}.\text{timestamp} \\
&\quad \wedge \text{SendMessage}(m, [type \mapsto \text{BackupRequest},
\end{aligned}$$

$$\begin{aligned}
& \text{did} \mapsto d, \\
& \text{bid} \mapsto b, \\
& \text{bucket} \mapsto \text{bucket}, \\
& \text{src} \mapsto n] \\
\wedge \text{ UNCHANGED } \langle \text{states}, \text{terms}, \text{masterships}, \text{backups}, \text{flows} \rangle
\end{aligned}$$

Handles a backup request 'm' on node 'n'

If the bucket contained in the backup request is more up-to-date than the same bucket on node 'n', node 'n's flows will be updated with the newer bucket and a successful response will be sent. Otherwise, a failed response will be sent.

$\text{HandleBackupRequest}(n, m) \triangleq$

IF

$$\begin{aligned}
& \wedge \text{states}[n][m.\text{did}].\text{term} = m.\text{term} \\
& \wedge \text{flows}[n][m.\text{did}][m.\text{bid}].\text{term} \leq m.\text{bucket}.\text{term} \\
& \wedge \text{flows}[n][m.\text{did}][m.\text{bid}].\text{timestamp} < m.\text{bucket}.\text{timestamp}
\end{aligned}$$

THEN

$$\begin{aligned}
& \wedge \text{flows}' = [\text{flows} \text{ EXCEPT } ![n][m.\text{did}][m.\text{bid}] = m.\text{bucket}] \\
& \wedge \text{SendMessage}(m.\text{src}, [\text{type} \mapsto \text{BackupResponse}, \\
& \quad \text{did} \mapsto m.\text{did}, \\
& \quad \text{bid} \mapsto m.\text{bid}, \\
& \quad \text{timestamp} \mapsto m.\text{bucket}.\text{timestamp}, \\
& \quad \text{succeeded} \mapsto \text{TRUE}]) \\
& \wedge \text{UNCHANGED } \langle \text{states}, \text{terms}, \text{masterships}, \text{backups} \rangle
\end{aligned}$$

ELSE

$$\begin{aligned}
& \wedge \text{SendMessage}(m.\text{src}, [\text{type} \mapsto \text{BackupResponse}, \text{succeeded} \mapsto \text{FALSE}]) \\
& \wedge \text{UNCHANGED } \langle \text{states}, \text{terms}, \text{masterships}, \text{backups}, \text{flows} \rangle
\end{aligned}$$

Handles a backup response 'm' on node 'n'

If the backup succeeded, the last backup timestamp on node 'n' is updated from the response.

Note the backup timestamp is sent in the response message. This is just a product of the language and not an actual implementation detail.

$\text{HandleBackupResponse}(n, m) \triangleq$

IF

$$\begin{aligned}
& \wedge m.\text{succeeded} \\
& \wedge \text{backups}[n][m.\text{did}][m.\text{bid}][m.\text{src}] < m.\text{timestamp}
\end{aligned}$$

THEN

$$\begin{aligned}
& \wedge \text{backups}' = [\text{backups} \text{ EXCEPT } ![n][m.\text{did}][m.\text{bid}][m.\text{src}] = m.\text{timestamp}] \\
& \wedge \text{UNCHANGED } \langle \text{states}, \text{terms}, \text{masterships}, \text{flows} \rangle
\end{aligned}$$

ELSE

$$\text{UNCHANGED } \langle \text{states}, \text{terms}, \text{masterships}, \text{backups}, \text{flows} \rangle$$

Sends a digest request for device 'd' from node 'n' to node 'm'.

The digest request is part of the anti-entropy protocol which requests bucket timestamps from a remote node 'm' to determine whether any flows are missing from the local node 'n'.

$\text{RequestDigests}(n, d, m) \triangleq$

$\wedge n \neq m$

$\wedge \text{SendMessage}(m, [type \mapsto \text{DigestsRequest},$
 $\quad \quad \quad did \mapsto d,$
 $\quad \quad \quad src \mapsto n])$
 $\wedge \text{UNCHANGED} \langle states, terms, masterships, backups, flows \rangle$

Handles a digest request 'm' on node 'n'

When the digest request is received, a function of buckets is returned containing the bucket digests.

Digests include the last term and logical time at which the bucket was updated on node 'n'.

$\text{HandleDigestsRequest}(n, m) \triangleq$
 $\wedge \text{LET } digests \triangleq [bucket \in \text{DOMAIN } flows[n][m.did] \mapsto [$
 $\quad \quad \quad term \mapsto flows[n][m.did][bucket].term,$
 $\quad \quad \quad timestamp \mapsto flows[n][m.did][bucket].timestamp]]$
 $\text{IN } \text{SendMessage}(m.src, [type \mapsto \text{DigestsResponse},$
 $\quad \quad \quad did \mapsto m.did,$
 $\quad \quad \quad src \mapsto n,$
 $\quad \quad \quad digests \mapsto digests])$
 $\wedge \text{UNCHANGED} \langle states, terms, masterships, backups, flows \rangle$

Handles a digest response 'm' on node 'n'

Digests are tuples of the term and timestamp at which the bucket was last updated.

This implementation defines a function for which the domain is buckets that are more up-to-date than on local node 'n' according to the digests.

$\text{HandleDigestsResponse}(n, m) \triangleq$
 $\wedge \text{LET } buckets \triangleq \{ bucket \in \text{DOMAIN } flows[n][m.did] :$
 $\quad \quad \quad \vee flows[n][m.did][bucket].term > m.digests[bucket].term$
 $\quad \quad \quad \vee flows[n][m.did][bucket].timestamp > m.digests[bucket].timestamp \}$
 $\text{IN } \text{SendMessage}(m.src, [type \mapsto \text{BucketsRequest},$
 $\quad \quad \quad did \mapsto m.did,$
 $\quad \quad \quad src \mapsto n,$
 $\quad \quad \quad buckets \mapsto buckets])$
 $\wedge \text{UNCHANGED} \langle states, terms, masterships, backups, flows \rangle$

Handles a bucket request 'm' on node 'n'

This implementation differs from the real-world implementation in that it handles an arbitrary number of buckets in the request and thus is not designed for scalability.

$\text{HandleBucketsRequest}(n, m) \triangleq$
 $\wedge \text{LET } buckets \triangleq [bucket \in m.buckets \mapsto flows[n][m.did][bucket]]$
 $\text{IN } \text{SendMessage}(m.src, [type \mapsto \text{BucketsResponse},$
 $\quad \quad \quad did \mapsto m.did,$
 $\quad \quad \quad src \mapsto n,$
 $\quad \quad \quad buckets \mapsto buckets])$
 $\wedge \text{UNCHANGED} \langle states, terms, masterships, backups, flows \rangle$

Handles a bucket response 'm' on node 'n'

This implementation differs from the real-world implementation in that it handles an arbitrary number of buckets in the response and thus is not designed for scalability.

$$\begin{aligned}
& \text{HandleBucketsResponse}(n, m) \triangleq \\
& \quad \wedge \text{flows}' = [\text{flows} \text{ EXCEPT } ![n][m.\text{did}] = [b \in 1 \dots \text{NumBuckets} \mapsto \\
& \quad \quad \text{IF } b \in \text{DOMAIN } m.\text{buckets} \text{ THEN } m.\text{buckets}[b] \text{ ELSE } \text{flows}[n][m.\text{did}][b]]] \\
& \quad \wedge \text{UNCHANGED } \langle \text{states}, \text{terms}, \text{backups}, \text{flows} \rangle
\end{aligned}$$

Handles a message 'm' on node 'n'

$$\begin{aligned}
& \text{HandleMessage}(n, m) \triangleq \\
& \quad \wedge \vee \wedge m.\text{type} = \text{BackupRequest} \\
& \quad \quad \wedge \text{HandleBackupRequest}(n, m) \\
& \quad \vee \wedge m.\text{type} = \text{BackupResponse} \\
& \quad \quad \wedge \text{HandleBackupResponse}(n, m) \\
& \quad \vee \wedge m.\text{type} = \text{DigestsRequest} \\
& \quad \quad \wedge \text{HandleDigestsRequest}(n, m) \\
& \quad \vee \wedge m.\text{type} = \text{DigestsResponse} \\
& \quad \quad \wedge \text{HandleDigestsResponse}(n, m) \\
& \quad \vee \wedge m.\text{type} = \text{BucketsRequest} \\
& \quad \quad \wedge \text{HandleBucketsRequest}(n, m) \\
& \quad \vee \wedge m.\text{type} = \text{BucketsResponse} \\
& \quad \quad \wedge \text{HandleBucketsResponse}(n, m) \\
& \quad \wedge \text{ReceiveMessage}(n, m)
\end{aligned}$$

$$\text{vars} \triangleq \langle \text{messages}, \text{states}, \text{backups}, \text{clocks}, \text{flows} \rangle$$

$$\begin{aligned}
& \text{Init} \triangleq \\
& \quad \wedge \text{messages} = [n \in \text{Node} \mapsto \langle \rangle] \\
& \quad \wedge \text{states} = [n \in \text{Node} \mapsto [d \in \text{Device} \mapsto [\text{term} \mapsto 0, \text{master} \mapsto \text{FALSE}]]] \\
& \quad \wedge \text{terms} = [d \in \text{Device} \mapsto 0] \\
& \quad \wedge \text{masterships} = [d \in \text{Device} \mapsto [n \in \text{Node} \mapsto \langle \rangle]] \\
& \quad \wedge \text{backups} = [n \in \text{Node} \mapsto [d \in \text{Device} \mapsto [b \in 1 \dots \text{NumBuckets} \mapsto 0]]] \\
& \quad \wedge \text{clocks} = [n \in \text{Node} \mapsto 0] \\
& \quad \wedge \text{flows} = [n \in \text{Node} \mapsto [d \in \text{Device} \mapsto [b \in 1 \dots \text{NumBuckets} \mapsto [\text{term} \mapsto 0, \text{timestamp} \mapsto 0, \text{entries} \mapsto \{\}]]]]
\end{aligned}$$

$$\begin{aligned}
& \text{Next} \triangleq \\
& \quad \vee \exists n \in \text{Node} : \exists f \in \text{Flow} : \text{AddFlow}(n, f) \\
& \quad \vee \exists n \in \text{Node} : \exists f \in \text{Flow} : \text{RemoveFlow}(n, f) \\
& \quad \vee \exists n \in \text{Node} : \exists m \in \text{Node} : \exists d \in \text{Device} : \exists b \in 1 \dots \text{NumBuckets} : \text{Backup}(n, d, b, m) \\
& \quad \vee \exists n \in \text{Node} : \exists m \in \text{Node} : \exists d \in \text{Device} : \text{RequestDigests}(n, d, m) \\
& \quad \vee \exists n \in \text{Node} : \exists d \in \text{Device} : \exists t \in 1 \dots \text{Nat} : \text{AddTerm}(n, d, t) \\
& \quad \vee \exists n \in \text{Node} : \exists d \in \text{Device} : \exists t \in \text{masterships}[d][n] : \text{LearnTerm}(n, d, t) \\
& \quad \vee \exists n \in \text{Node} : \exists m \in \text{messages}[n] : \text{HandleMessage}(n, m)
\end{aligned}$$

$$\text{Spec} \triangleq \text{Init} \wedge \square[\text{Next}]_{\text{vars}}$$

\ * Modification History

* Last modified *Wed Jun 20 17:49:54 PDT 2018* by *jordanhalterman*
* Created *Mon Jun 18 21:52:20 PDT 2018* by *jordanhalterman*